

# Introduction to Boosted Trees

---

Tianqi Chen  
Oct. 22 2014

# Outline

---

- Review of key concepts of supervised learning
  - Regression Tree and Ensemble (What are we Learning)
  - Gradient Boosting (How do we Learn)
  - Summary
-

# Elements in Supervised Learning

---

- Notations:  $x_i \in \mathbb{R}^d$  i-th training example
  - **Model:** how to make prediction  $\hat{y}_i$  given  $x_i$ 
    - Linear model:  $\hat{y}_i = \sum_j w_j x_{ij}$  (include linear/logistic regression)
    - The prediction score  $\hat{y}_i$  can have different interpretations depending on the task
      - ♦ Linear regression:  $\hat{y}_i$  is the predicted score
      - ♦ Logistic regression:  $1/(1 + \exp(-\hat{y}_i))$  is predicted the probability of the instance being positive
      - ♦ Others... for example in ranking  $\hat{y}_i$  can be the rank score
  - **Parameters:** the things we need to learn from data
    - Linear model:  $\Theta = \{w_j | j = 1, \dots, d\}$
-

# Elements continued: Objective Function

---

- Objective function that is everywhere

$$Obj(\Theta) = L(\Theta) + \Omega(\Theta)$$

**Training Loss** measures how well model fit on training data

**Regularization**, measures complexity of model

- Loss on training data:  $L = \sum_{i=1}^n l(y_i, \hat{y}_i)$ 
    - Square loss:  $l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$
    - Logistic loss:  $l(y_i, \hat{y}_i) = y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i})$
  - Regularization: how complicated the model is?
    - L2 norm:  $\Omega(w) = \lambda \|w\|^2$
    - L1 norm (lasso):  $\Omega(w) = \lambda \|w\|_1$
-

# Putting known knowledge into context

---

- Ridge regression:  $\sum_{i=1}^n (y_i - w^T x_i)^2 + \lambda \|w\|^2$ 
    - Linear model, square loss, L2 regularization
  - Lasso:  $\sum_{i=1}^n (y_i - w^T x_i)^2 + \lambda \|w\|_1$ 
    - Linear model, square loss, L1 regularization
  - Logistic regression:
$$\sum_{i=1}^n [y_i \ln(1 + e^{-w^T x_i}) + (1 - y_i) \ln(1 + e^{w^T x_i})] + \lambda \|w\|^2$$
    - Linear model, logistic loss, L2 regularization
  - The conceptual separation between model, parameter, objective also gives you **engineering benefits**.
    - Think of how you can implement SGD for both ridge regression and logistic regression
-

# Objective and Bias Variance Trade-off

---

$$Obj(\Theta) = L(\Theta) + \Omega(\Theta)$$

**Training Loss** measures how well model fit on training data

**Regularization**, measures complexity of model

- Why do we want to contain two component in the objective?
  - Optimizing training loss encourages **predictive** models
    - Fitting well in training data at least get you close to training data which is hopefully close to the underlying distribution
  - Optimizing regularization encourages **simple** models
    - Simpler models tends to have smaller variance in future predictions, making prediction **stable**
-

# Outline

---

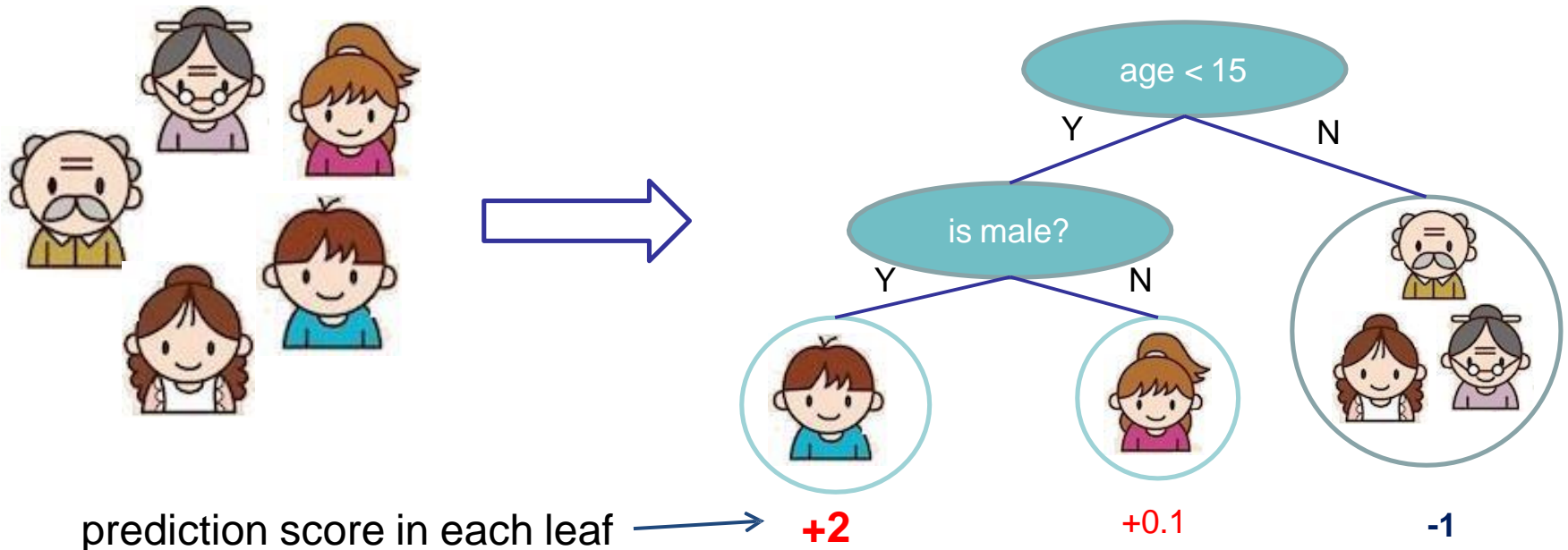
- Review of key concepts of supervised learning
  - **Regression Tree and Ensemble (What are we Learning)**
  - Gradient Boosting (How do we Learn)
  - Summary
-

# Regression Tree (CART)

- regression tree (also known as classification and regression tree):
  - Decision rules same as in decision tree
  - Contains one score in each leaf value

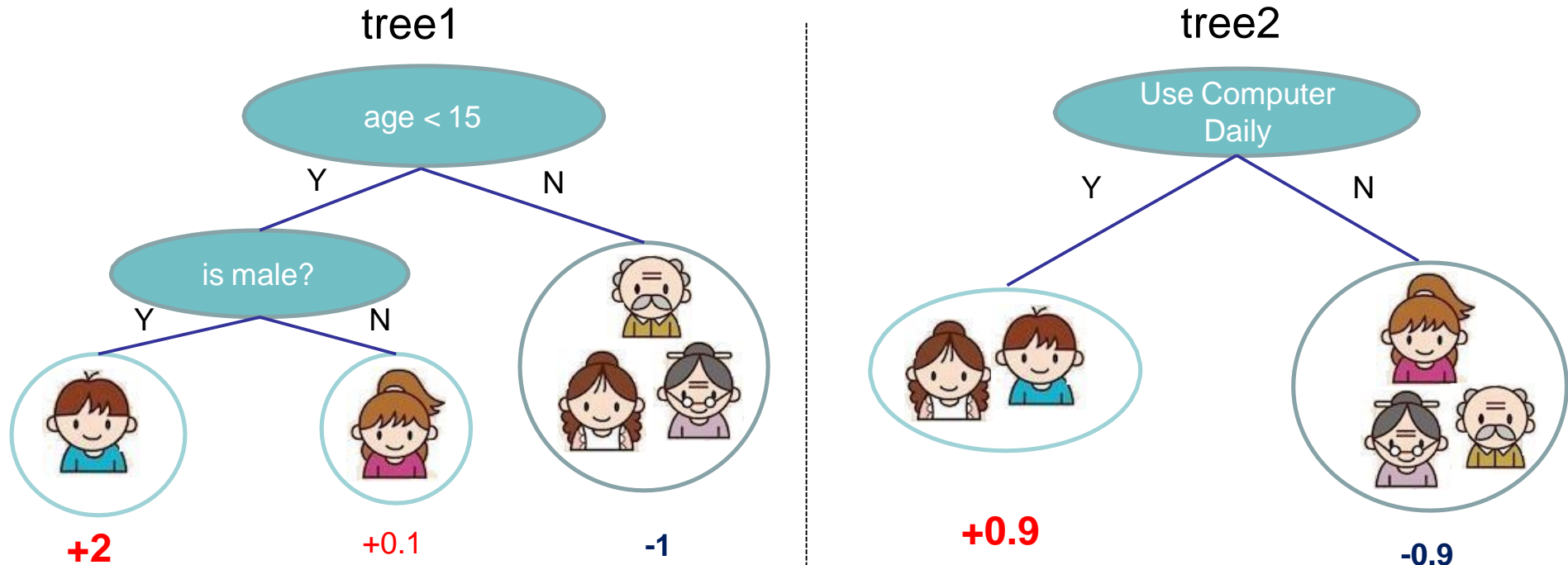
Input: age, gender, occupation, ...

Does the person like computer games





# Regression Tree Ensemble



$$f(\text{boy}) = 2 + 0.9 = 2.9$$

$$f(\text{old man}) = -1 - 0.9 = -1.9$$

Prediction of is sum of scores predicted by each of the tree

# Tree Ensemble methods

---

- Very widely used, look for GBM, random forest...
    - Almost half of data mining competition are won by using some variants of tree ensemble methods
  - Invariant to scaling of inputs, so you do not need to do careful features normalization.
  - Learn higher order interaction between features.
  - Can be scalable, and are used in Industry
-

# Put into context: Model and Parameters

---

- Model: assuming we have K trees

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i), \quad f_k \in \mathcal{F}$$

Space of functions containing all Regression trees

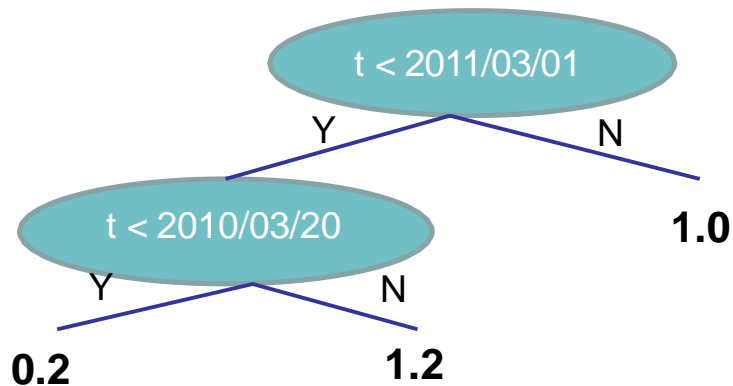
*Think: regression tree is a function that maps the attributes to the score*

- Parameters
    - Including structure of each tree, and the score in the leaf
    - Or simply use function as parameters
$$\Theta = \{f_1, f_2, \dots, f_K\}$$
    - Instead learning weights in  $\mathbf{R}^d$ , we are learning functions(trees)
-

# Learning a tree on single variable

- How can we learn functions?
- Define objective (loss, regularization), and optimize it!!
- Example:
  - Consider regression tree on single input  $t$  (time)
  - I want to predict whether I like romantic music at time  $t$

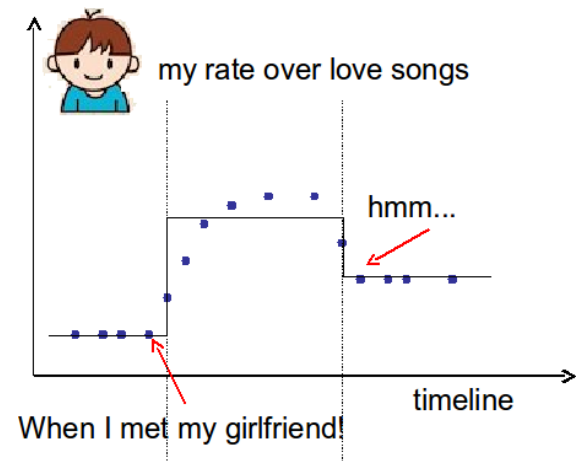
The model is regression tree that splits on time



Equivalently



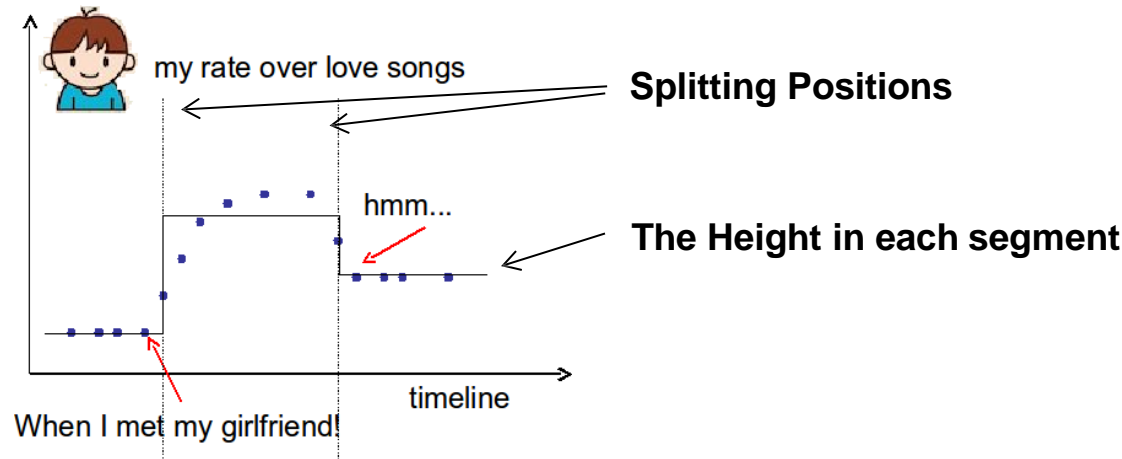
Piecewise step function over time



# Learning a step function

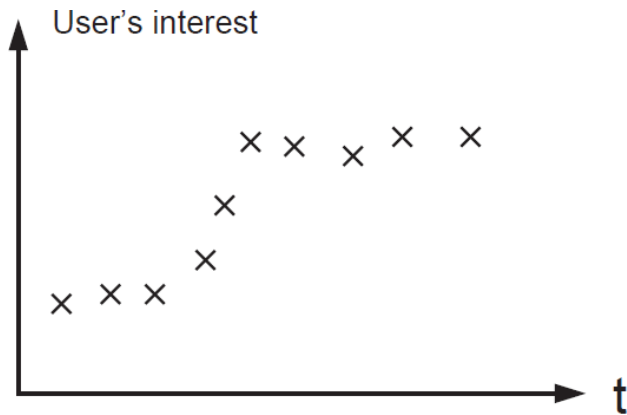
---

- Things we need to learn

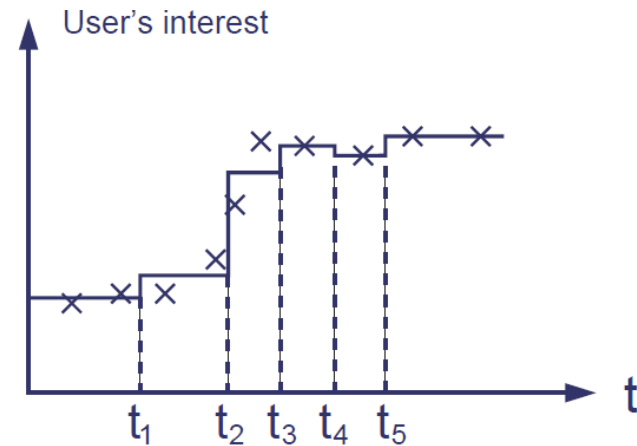


- Objective for single variable regression tree(step functions)
    - Training Loss: How will the function fit on the points?
    - Regularization: How do we define complexity of the function?
      - ◆ Number of splitting points,  $l_2$  norm of the height in each segment?
-

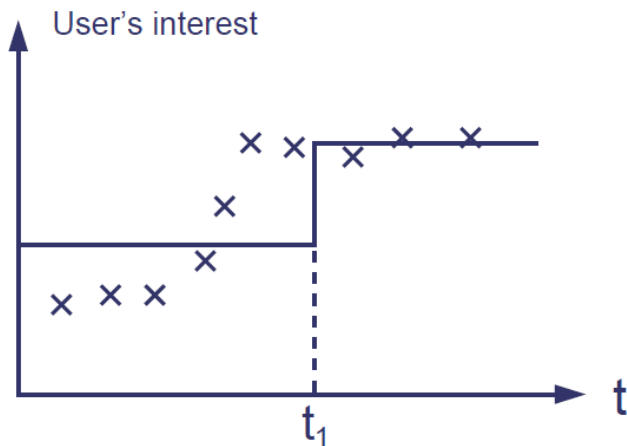
# Learning step function (visually)



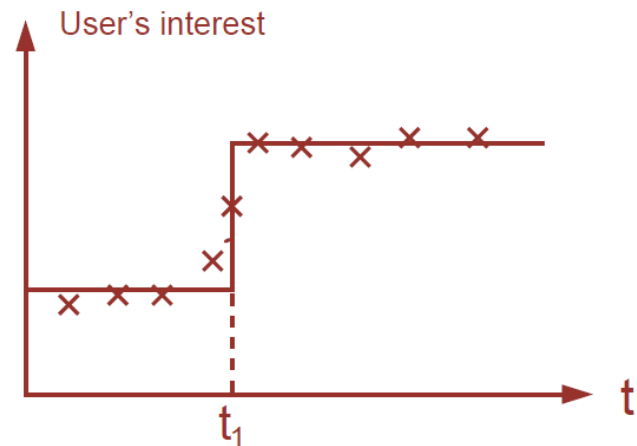
Observed user's interest on topic k against time t



☒ Too many splits,  $\Omega(f)$  is high



☒ Wrong split point,  $L(f)$  is high



☒ Good balance of  $\Omega(f)$  and  $L(f)$

# Coming back: Objective for Tree Ensemble

---

- Model: assuming we have K trees

$$\hat{y}_i = \sum_{k=1}^K f_k(x_i), \quad f_k \in \mathcal{F}$$

- Objective

$$Obj = \sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_{k=1}^K \Omega(f_k)$$

Training loss

Complexity of the Trees

- Possible ways to define  $\Omega$  ?
    - Number of nodes in the tree, depth
    - L2 norm of the leaf weights
    - ...detailed later
-

# Objective vs Heuristic

---

- When you talk about (decision) trees, it is usually heuristics
    - Split by information gain
    - Prune the tree
    - Maximum depth
    - Smooth the leaf values
  - Most heuristics maps well to objectives, taking the formal (objective) view let us know what we are learning
    - Information gain -> training loss
    - Pruning -> regularization defined by #nodes
    - Max depth -> constraint on the function space
    - Smoothing leaf values -> L2 regularization on leaf weights
-



# Regression Tree is not just for regression!

---

- Regression tree ensemble defines how you make the prediction score, it can be used for
    - Classification, Regression, Ranking...
    - ...
  - It all depends on how you define the objective function!
  - So far we have learned:
    - Using Square loss  $l(y_i, \hat{y}_i) = (y_i - \hat{y}_i)^2$ 
      - ♦ Will result in common gradient boosted machine
    - Using Logistic loss  $l(y_i, \hat{y}_i) = y_i \ln(1 + e^{-\hat{y}_i}) + (1 - y_i) \ln(1 + e^{\hat{y}_i})$ 
      - ♦ Will result in LogitBoost
-

# Outline

---

- Review of key concepts of supervised learning
  - Regression Tree and Ensemble (What are we Learning)
  - **Gradient Boosting (How do we Learn)**
  - Summary
-

# Take Home Message for this section

---

- Bias-variance tradeoff is everywhere
  - The loss + regularization objective pattern applies for regression tree learning (function learning)
  - We want **predictive** and **simple** functions
  - This defines what we want to learn (objective, model).
  - But how do we learn it?
    - Next section
-

# So How do we Learn?

---

- Objective:  $\sum_{i=1}^n l(y_i, \hat{y}_i) + \sum_k \Omega(f_k), f_k \in \mathcal{F}$
- We can not use methods such as SGD, to find  $f$  (since they are trees, instead of just numerical vectors)
- Solution: **Additive Training (Boosting)**
  - Start from constant prediction, add a new function each time

$$\hat{y}_i^{(0)} = 0$$

$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$

$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$

...

$$\hat{y}_i^{(t)} = \sum_{k=1}^t f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

← New function

Model at training round  $t$

Keep functions added in previous round

# Additive Training

---

- How do we decide which  $f$  to add?
  - Optimize the objective!!

- The prediction at round  $t$  is  $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$

 This is what we need to decide in round  $t$

$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^n l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^t \Omega(f_i) \\ &= \sum_{i=1}^n l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant \end{aligned}$$

 Goal: find  $f_t$  to minimize this

- Consider square loss

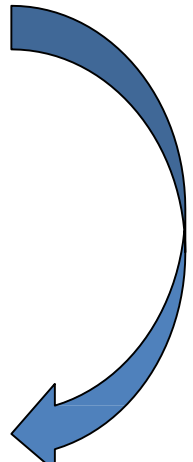
$$\begin{aligned} Obj^{(t)} &= \sum_{i=1}^n \left( y_i - (\hat{y}_i^{(t-1)} + f_t(x_i)) \right)^2 + \Omega(f_t) + const \\ &= \sum_{i=1}^n \left[ 2(\hat{y}_i^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2 \right] + \Omega(f_t) + const \end{aligned}$$

 This is usually called residual from previous round

---

# Taylor Expansion Approximation of Loss

---

- Goal  $Obj^{(t)} = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t) + constant$ 
    - Seems still complicated except for the case of square loss
  - Take Taylor expansion of the objective
    - Recall  $f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2$
    - Define  $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ ,  $h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}^{(t-1)})$
- 

$$Obj^{(t)} \simeq \sum_{i=1}^n \left[ l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) + constant$$

- *If you are not comfortable with this, think of square loss*

$$g_i = \partial_{\hat{y}^{(t-1)}} (\hat{y}^{(t-1)} - y_i)^2 = 2(\hat{y}^{(t-1)} - y_i) \quad h_i = \partial_{\hat{y}^{(t-1)}}^2 (\hat{y}^{(t-1)} - y_i)^2 = 2$$

- Compare what we get to previous slide
-

# Our New Goal

---

- Objective, with constants removed

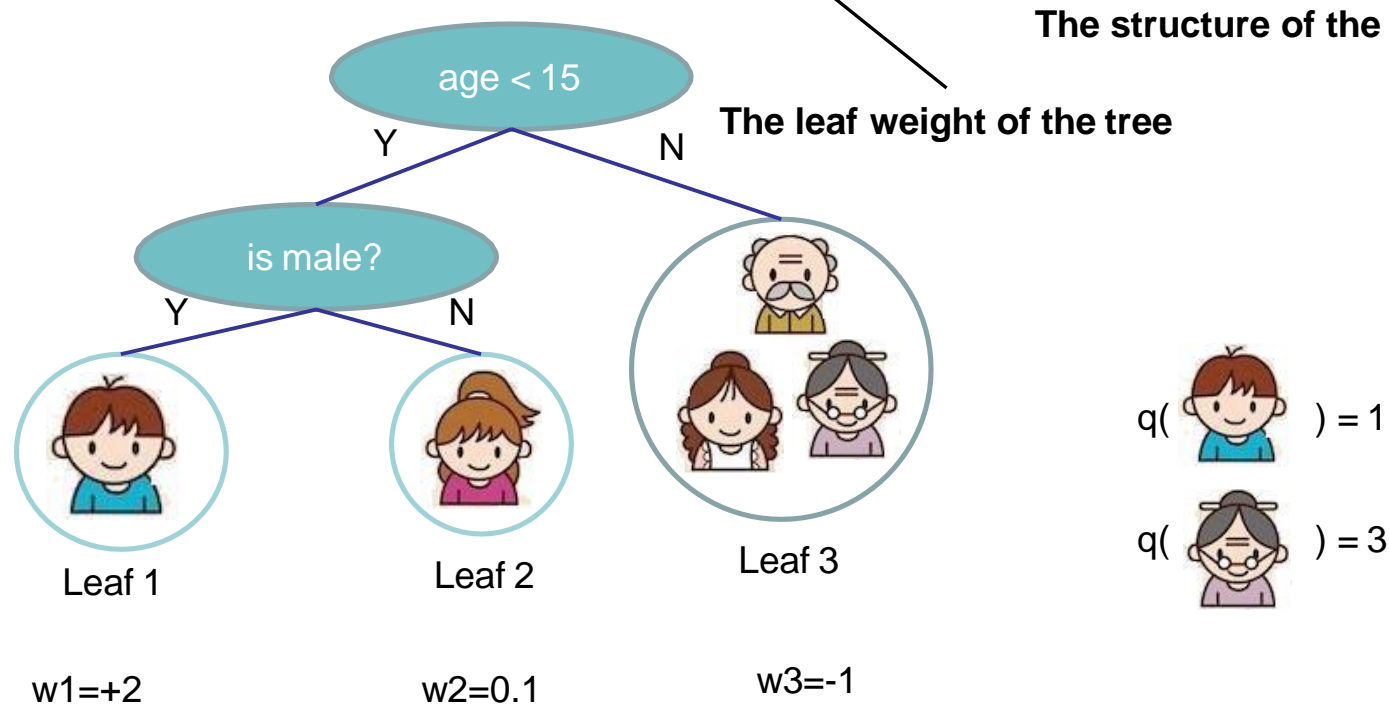
$$\sum_{i=1}^n \left[ g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

- where  $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$ ,  $h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}^{(t-1)})$
  - Why spending smuch efforts to derive the objective, why not just grow trees ...
    - Theoretical benefit: know what we are learning, convergence
    - **Engineering** benefit, recall the elements of supervised learning
      - ♦  $g_i$  and  $h_i$  comes from definition of loss function
      - ♦ The learning of function only depend on the objective via  $g_i$  and  $h_i$
      - ♦ Think of how you can separate modules of your code when you are asked to implement boosted tree for both square loss and logistic loss
-

# Refine the definition of tree

- We define tree by a vector of scores in leafs, and a leaf index mapping function that maps an instance to a leaf

$$f_t(x) = w_{q(x)}, \quad w \in \mathbf{R}^T, q : \mathbf{R}^d \rightarrow \{1, 2, \dots, T\}$$





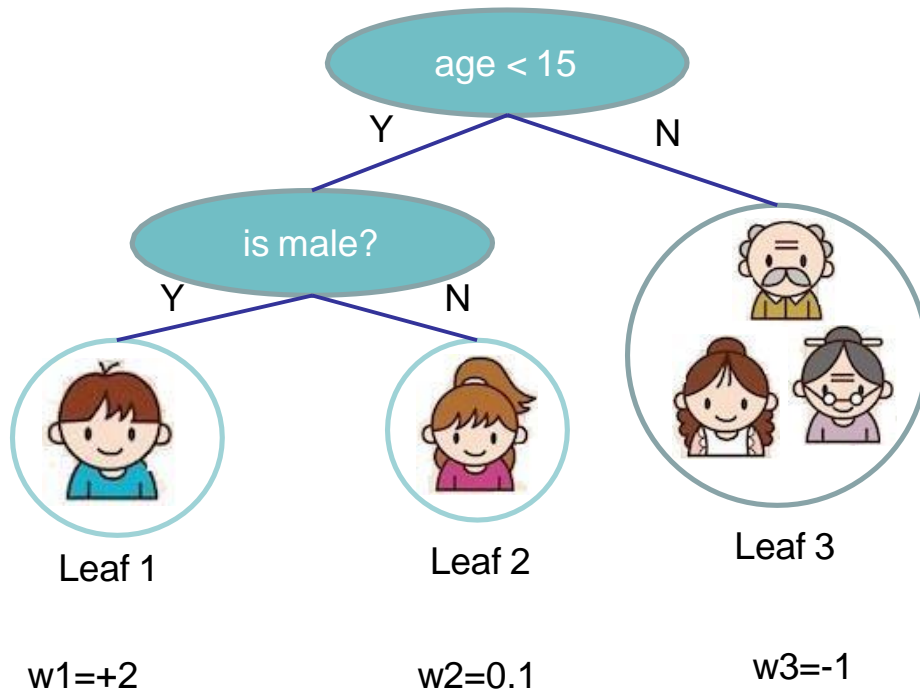
# Define Complexity of a Tree (cont')

- Define complexity as (this is not the only possible definition)

$$\Omega(f_t) = \gamma T + \frac{1}{2} \lambda \sum_{j=1}^T w_j^2$$

Number of leaves

L2 norm of leaf scores



$$\Omega = \gamma 3 + \frac{1}{2} \lambda (4 + 0.01 + 1)$$

# Revisit the Objectives

---

- Define the instance set in leaf  $j$  as  $I_j = \{i | q(x_i) = j\}$
- Regroup the objective by each leaf

$$\begin{aligned} Obj^{(t)} &\simeq \sum_{i=1}^n \left[ g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \\ &= \sum_{i=1}^n \left[ g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^T w_j^2 \\ &= \sum_{j=1}^T \left[ \left( \sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left( \sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T \end{aligned}$$

- This is sum of  $T$  independent quadratic functions
-

# The Structure Score

---

- Two facts about single variable quadratic function

$$\operatorname{argmin}_x Gx + \frac{1}{2}Hx^2 = -\frac{G}{H}, \quad H > 0 \quad \min_x Gx + \frac{1}{2}Hx^2 = -\frac{1}{2} \frac{G^2}{H}$$

- Let us define  $G_j = \sum_{i \in I_j} g_i$   $H_j = \sum_{i \in I_j} h_i$

$$\begin{aligned} \operatorname{Obj}^{(t)} &= \sum_{j=1}^T \left[ \left( \sum_{i \in I_j} g_i \right) w_j + \frac{1}{2} \left( \sum_{i \in I_j} h_i + \lambda \right) w_j^2 \right] + \gamma T \\ &= \sum_{j=1}^T \left[ G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T \end{aligned}$$

- Assume the structure of tree (  $q(x)$  ) is fixed, the optimal weight in each leaf, and the resulting objective value are

$$w_j^* = -\frac{G_j}{H_j + \lambda} \quad \operatorname{Obj} = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

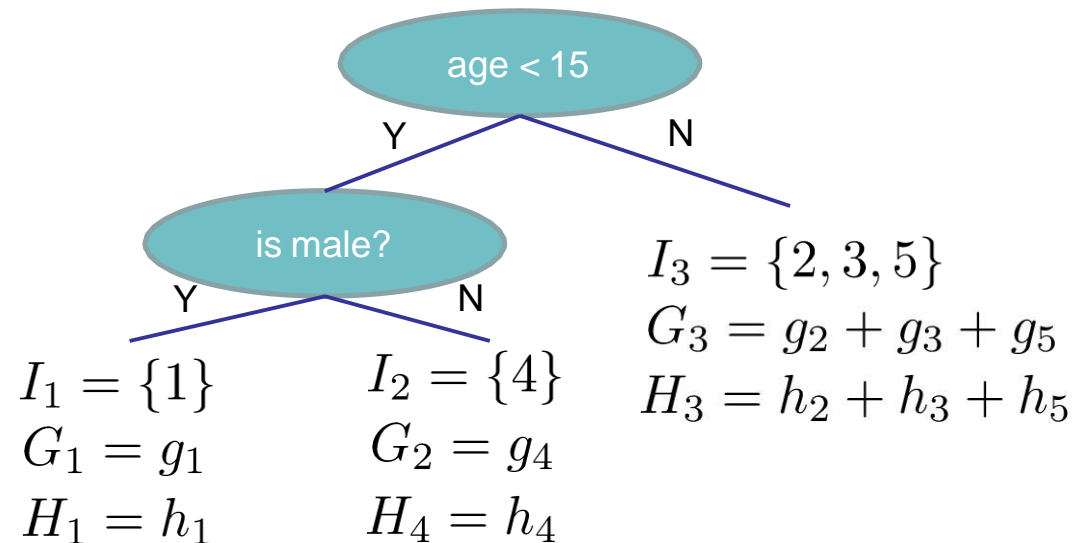
 This measures how good a tree structure is!

---

# The Structure Score Calculation

Instance index      gradient statistics

1		$g_1, h_1$
2		$g_2, h_2$
3		$g_3, h_3$
4		$g_4, h_4$
5		$g_5, h_5$



$$Obj = - \sum_j \frac{G_j^2}{H_j + \lambda} + 3\gamma$$

The smaller the score is, the better the structure is

# Searching Algorithm for Single Tree

---

- Enumerate the possible tree structures  $q$
- Calculate the structure score for the  $q$ , using the scoring eq.

$$Obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

- Find the best tree structure, and use the optimal leaf weight

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

- But...there can be infinite possible tree structures..
-

# Greedy Learning of the Tree

---

- In practice, we grow the tree greedily
  - Start from tree with depth 0
  - For each leaf node of the tree, try to add a split. The change of objective after adding the split is

$$Gain = \frac{1}{2} \left[ \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma$$

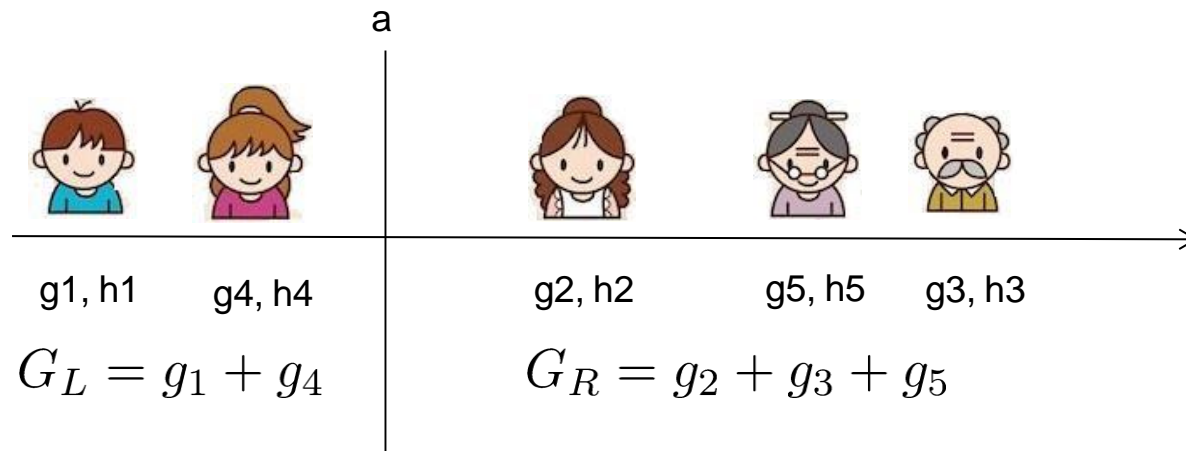
Annotations for the equation:

- the score of left child (points to  $\frac{G_L^2}{H_L + \lambda}$ )
- the score of right child (points to  $\frac{G_R^2}{H_R + \lambda}$ )
- the score of if we do not split (points to  $\frac{(G_L + G_R)^2}{H_L + H_R + \lambda}$ )
- The complexity cost by introducing additional leaf (points to  $\gamma$ )

- Remaining question: how do we find the best split?
-

# Efficient Finding of the Best Split

- What is the gain of a split rule  $x_j < a$  ? Say  $x_j$  is age



- All we need is sum of  $g$  and  $h$  in each side, and calculate

$$Gain = \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \gamma$$

- Left to right linear scan over sorted instance is enough to decide the best split along the feature

# An Algorithm for Split Finding

---

- For each node, enumerate over all features
    - For each feature, sorted the instances by feature value
    - Use a linear scan to decide the best split along that feature
    - Take the best split solution along all the features
  - Time Complexity growing a tree of depth  $K$ 
    - It is  $O(n d K \log n)$ : or each level, need  $O(n \log n)$  time to sort  
There are  $d$  features, and we need to do it for  $K$  level
    - This can be further optimized (e.g. use approximation or caching the sorted features)
    - Can scale to very large dataset
-



# What about Categorical Variables?

---

- Some tree learning algorithm handles categorical variable and continuous variable separately
  - We can easily use the scoring formula we derived to score split based on categorical variables.
- Actually it is not necessary to handle **categorical** separately.
  - We can encode the categorical variables into numerical vector using one-hot encoding. Allocate a #categorical length vector

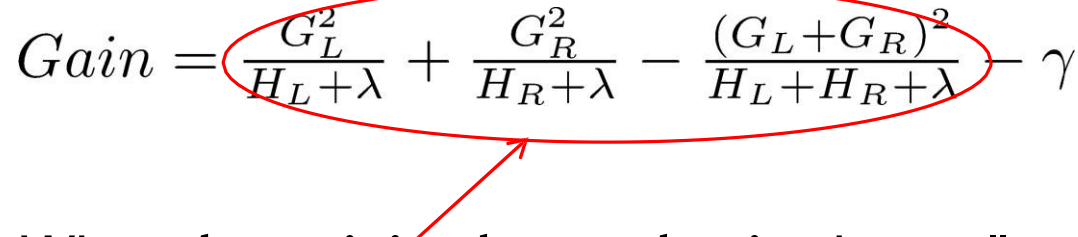
$$z_j = \begin{cases} 1 & \text{if } x \text{ is in category } j \\ 0 & \text{otherwise} \end{cases}$$

- The vector will be sparse if there are lots of categories, the learning algorithm is preferred to handle sparse data
-

# Pruning and Regularization

---

- Recall the gain of split, it can be **negative!**

$$Gain = \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \gamma$$


- When the training loss reduction is smaller than regularization
    - Trade-off between simplicity and predictiveness
  - Pre-stopping
    - Stop split if the best split have negative gain
    - But maybe a split can benefit future splits..
  - Post-Pruning
    - Grow a tree to maximum depth, recursively prune all the leaf splits with negative gain
-

# Recap: Boosted Tree Algorithm

---

- Add a new tree in each iteration
- Beginning of each iteration, calculate

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial_{\hat{y}^{(t-1)}}^2 l(y_i, \hat{y}^{(t-1)})$$

- Use the statistics to greedily grow a tree  $f_t(x)$

$$Obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

- Add  $f_t(x)$  to the model  $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$ 
    - Usually, instead we do  $y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$
    - $\epsilon$  is called step-size or shrinkage, usually set around 0.1
    - This means we do not do full optimization in each step and reserve chance for future rounds, it helps prevent overfitting
-

# Outline

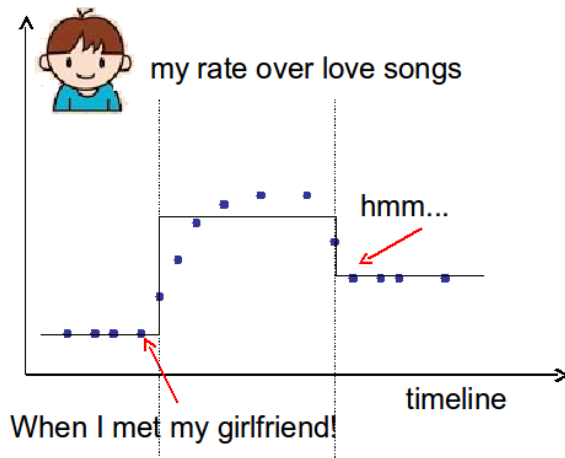
---

- Review of key concepts of supervised learning
  - Regression Tree and Ensemble (What are we Learning)
  - Gradient Boosting (How do we Learn)
  - **Summary**
-

# Questions to check if you really get it

---

- How can we build a boosted tree classifier to do weighted regression problem, such that each instance have a importance weight?
- Back to the time series problem, if I want to learn step functions over time. Is there other ways to learn the time splits, other than the top down split approach?



# Questions to check if you really get it

---

- How can we build a boosted tree classifier to do weighted regression problem, such that each instance have a importance weight?
  - Define objective, calculate  $g_i, h_i$ , feed it to the old tree learning algorithm we have for un-weighted version

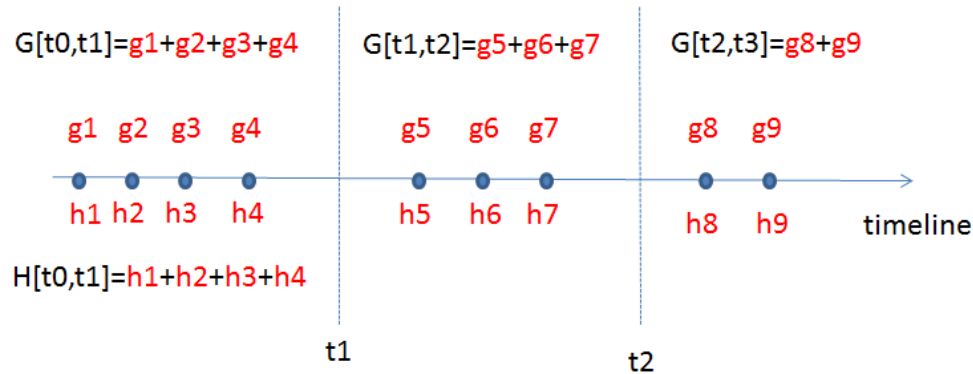
$$l(y_i, \hat{y}_i) = \frac{1}{2} a_i (\hat{y}_i - y_i)^2 \quad g_i = a_i (\hat{y}_i - y_i) \quad h_i = a_i$$

- Again think of separation of model and objective, how does the theory can help better organizing the machine learning toolkit
-

# Questions to check if you really get it

---

- Time series problem



- All that is important is the structure score of the splits

$$Obj = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T$$

- Top-down greedy, same as trees
  - Bottom-up greedy, start from individual points as each group, greedily merge neighbors
  - Dynamic programming, can find optimal solution for this case
-

# Summary

---

- The separation between model, objective, parameters can be helpful for us to understand and customize learning models
- The bias-variance trade-off applies everywhere, including learning in functional space

$$Obj(\Theta) = L(\Theta) + \Omega(\Theta)$$

- We can be formal about what we learn and how we learn. Clear understanding of theory can be used to guide cleaner implementation.
-



# Reference

---

- Greedy function approximation a gradient boosting machine. *J.H. Friedman*
    - *First paper about gradient boosting*
  - *Stochastic Gradient Boosting. J.H. Friedman*
    - *Introducing bagging trick to gradient boosting*
  - *Elements of Statistical Learning. T. Hastie, R. Tibshirani and J.H. Friedman*
    - *Contains a chapter about gradient boosted boosting*
  - Additive logistic regression a statistical view of boosting. *J.H. Friedman T. Hastie R. Tibshirani*
    - *Uses second-order statistics for tree splitting, which is closer to the view presented in this slide*
  - Learning Nonlinear Functions Using Regularized Greedy Forest. *R. Johnson and T. Zhang*
    - *Proposes to do fully corrective step, as well as regularizing the tree complexity. The regularizing trick is closed related to the view present in this slide*
  - Software implementing the model described in this slide: <https://github.com/tqchen/xgboost>
-